51CTO.com
技术成就梦想

我 们 只 谈 开 发

开发月刊

Development Monthly

2013年02月
总第023期

Python：十年语言之冠

Python简史

为何MongoDB是当之无愧NoSQL王者

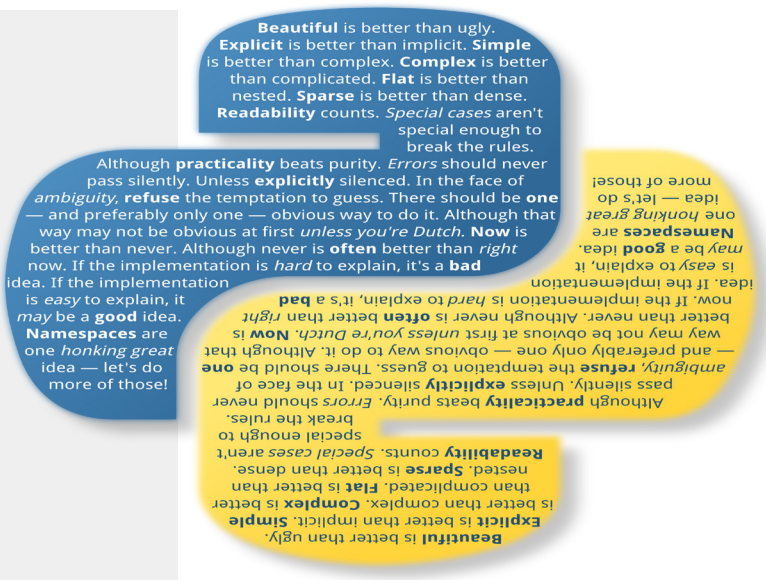
目录

2013.02

Contents

51CTO.com
技术成就梦想

出版方 :北京无忧创想信息技术有限公司
责任编辑 :彭凡　张伟　林师授　　封面设计 :苍旭
联系方法 :zhangwei@51cto.com　010-68476606 (分机 8110)
出版日期 :2013 年 2 月 26 日
欢迎广大读者投稿



python™

Python 最近异常的火热，这门语言又被国外评为十年语言之冠。

我们都知道，Python 起源于一种脚本语言，用于科学计算，现在它日渐流行，不断向 Web 和商业等技术领域渗透。这些技术领域都由一些大公司(Java，C#，C++) 在背后推断，有大量的资金支持，而在 web 方面，PHP 和 Ruby 也名噪一时。这这样大的竞争压力下，一种新语言想流行起来是非常困难的。Python 成功了。对我来说，这就是为什么 Python 是十年语言之首。

最初的 Python logo: 由 Guido 的兄弟 Just von Rossum 设计

Python 从一开始就特别在意可拓展性 (extensibility)。 Python 可以在多个层次上拓展。从高层上，你可以引入 .py 文件。

在底层，你可以引用 C 语言的库。Python 程序员可以快速的使用 Python 写 .py 文件作为拓展模块。

但当性能是考虑的重要因素时，Python 程序员可以深入底层，写 C 程序，编译为 .so 文件引入到 Python 中使用。

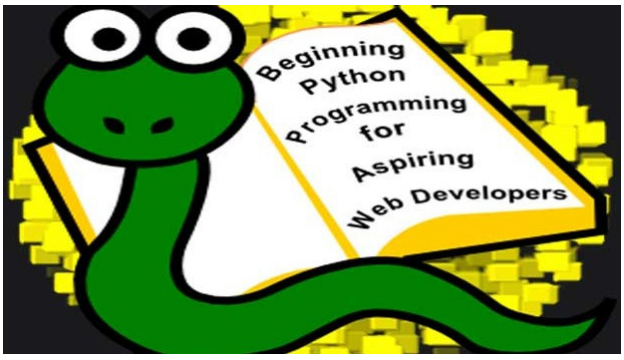
Python 就好像是使用钢构建房一样，先规定好大的框架。而程序员可以在此框架下相当自由的拓展或更改。

51CTO 开发频道语

编者按

通过作者的研究，我们可以得知，在过去的十年里，Python 语言获得了最大的增长幅度。从图片上我们可以看到 10 年内它的非常漂亮的线性增长，恭贺 Python 语言！

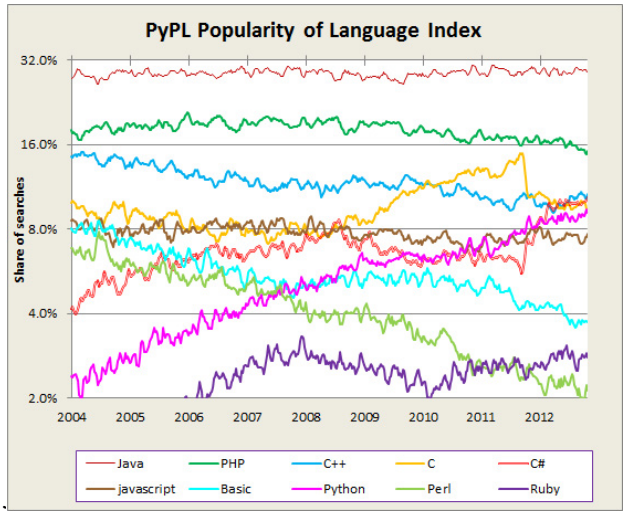
Python：十年语言之冠



最近我发现了这个 PYPL——编程语言流行指数。它对各种语言的流行指标进行了二次发掘。作者指出 TIOBE 指数很可能不能反映出真实情况，归咎于一些编程语言的名称会导致误解。

他引入了一些新术语，利用谷歌趋势得出来不同的结论。

通过作者的研究，我们可以得知，在过去的十年里，Python 语言获得了最大的增长幅度。从图片上我们可以看到 10 年内它的非常漂亮的线性增长，恭贺 Python 语言！



Python 是唯一一个在这个图表上表现的与众不同的语言。

我们都知道，Python 起源于一种脚本语言，用于科学计算，现在它日渐流行，不断向 Web 和商业等技术领域渗透。这些技术领域都由一些大公司 (Java，C#，C++) 在背后推断，有大量的资金支持，而在 web 方面，PHP 和 Ruby 也名噪一时。这这样大的竞争压力下，一种新语言想流行起来是非常困难的。Python 成功了。对我来说，这就是为什么 Python 是十年语言之首。

英文原文 :Python – language of the decade

不知从何时起，“压力山大” 成了 IT 人士自嘲的流行语。很多 IT 人士在工作与健康之间，以生命的逝去为一个感叹号，生活的天枰已然失衡。过渡的劳累以及对病情的耽搁，最终将美好的生活吞噬殆尽。一句话来评价 IT 人的工作的：把男人当超人用，把女人当男人用，这就是 IT 人的一个工作情况。

专题链接：

http://developer.51cto.com/exp/forever/

Python简史

Python是我喜欢的语言，简洁，优美，容易使用。前两天，我很激昂向朋友宣传Python的好处。

听过之后，朋友问我：好吧，我承认Python不错，但它为什么叫Python呢？

我不是很确定：呃，似乎是个电视剧的名字。

朋友又问：那你说的Guido是美国人么？(Guido von Rossum，Python的作者)

我再次不是很确定：他从google换到Dropbox工作，但他的名字像是荷兰人的（有一个von在中间）。

所以，后面我花了些时间调查Python的历史。这是很好的学习。我看到了Python中许多功能的来源和Python的设计理念，比如哪些功能是历史遗留，哪些功能是重复，如何增加功能……而且，Python也是开源(open source)运动的一个成功案例。从Python的历史中，我们可以一窥开源开发的理念和成就。

这也可以作为我写的Python快速教程序篇。

Python的起源

Python的作者，Guido von Rossum，确实是荷兰人。1982年，Guido从阿姆斯特丹大学(University of Amsterdam)获得了数学和计算机硕士学位。然而，尽管他算得上是一位数学家，他更加享受计算机带来的乐趣。用他的话说，尽管拥有数学和计算机双料资质，他总趋向于做计算机相关工作，并热衷于做任何和编程相关的活儿。

在那个时候，他接触并使用过诸如Pascal、C、Fortran等语言。这些语言的基本设计原则是让机器能更快运行。在80年代，虽然IBM和苹果已经掀起了个人电脑浪潮，但这些个人电脑的配置很低（在今天看来）。比如早期的Macintosh，只有8MHz的CPU主频和128KB的RAM，一个大的数组就能占满内存。所有的编译器的核心是做优化，以便让程序能够运行。为了增进效率，语言也迫使程序员像计算机一样思考，以便能写出更符合机器口味的程序。在那个时代，程序员恨不得用手榨取计算机每一寸的能力。有人甚至认为C语言的指针是在浪费内存。至于动态类型，内存自动管理，面向对象…… 别想了，那会让你的电脑陷入瘫痪。

然而，这种思考方式让Guido感到苦恼。Guido知道如何用C语言写出一个功能，但整个编写过程需要耗费大量的时间（即使他已经准确的知道了如何实现）。他的另一个选择是shell。Bourne Shell作为UNIX系统的解释器(interpreter)已经长期存在。

UNIX的管理员们常常用shell去写一些简单的脚本，以进行一些系统维护的工作，比如定期备份、文件系统管理等等。shell可以像胶水一样，将UNIX下的许多功能连接在一起。许多C语言下上百行的程序，在shell下只用几行就可以完成。

然而，shell的本质是调用命令。它并不是一个真正的语言。比如说，shell没有数值型的数据类型，加法运算都很复杂。总之，shell不能全面的调动计算机的功能。

Python 简史 II

(关于 shell,你可以参考 Linux 架构和 Linux 命令行与命令)

Guido 希望有一种语言,这种语言能够像 C 语言那样,能够全面调用计算机的功能接口,又可以像 shell 那样,可以轻松的编程。ABC 语言让 Guido 看到希望。ABC 是由荷兰的 CWI (Centrum Wiskunde & Informatica, 数学和计算机研究所) 开发的。Guido 在 CWI 工作,并参与到 ABC 语言的开发。ABC 语言以教学为目的。与当时的大部分语言不同, ABC 语言的目标是“让用户感觉更好”。ABC 语言希望让语言变得容易阅读,容易使用,容易记忆,容易学习,并以此来激发人们学习编程的兴趣。比如下面是一段来自 Wikipedia 的 ABC 程序,这个程序用于统计文本中出现的词 (word) 的总数 :

```
HOW TO RETURN words document:
  PUT {} IN collection
  FOR line IN document:
    FOR word IN split line:
      IF word not.in collection:
        INSERT word IN collection
  RETURN collection
```

HOW TO 用于定义一个函数。一个 Python 程序员应该很容易理解这段程序。ABC 语言使用冒号 (:) 和缩进来表示程序块 (C 语言使用 {} 来表示程序块)。行 尾没有分号。for 和 if 结构中也没有括号 ()。

如果将 HOW TO 改为 def, 将 PUT 行改为 collection = [], 将 INSERT 行改为 collection.append(word),这就几乎是一个标准的 Python 函数。上面的函数读起来就像一段自然的文字。

尽管已经具备了良好的可读性和易用性, ABC 语言最终没有流行起来。在当时, ABC 语言编译器需要比较高配置的电脑才能运行。

而这些电脑的使用者通常精通计算机,他们更多考虑程序的效率,而非它的学习难度。除了硬件上的困难外, ABC 语言的设计也存在一些致命的问题 :

· 可拓展性差。ABC 语言不是模块化语言。如果想在 ABC 语言中增加功能,比如对图形化的支持,就必须改动很多地方。

· 不能直接进行 IO。ABC 语言不能直接操作文件系统。尽管你可以通过诸如文本流的方式导入数据,但 ABC 无法直接读写文件。输入输出的困难对于计算机语言来说是致命的。你能想像一个打不开车门的跑车么?

· 过度革新。 ABC 用自然语言的方式来表达程序的意义,比如上面程序中的 HOW TO (如何)。然而对于程序员来说,他们更习惯用 function 或者 define 来定义一个函数。同样,程序员也习惯了用等号 (=) 来分配变量。这尽管让 ABC 语言显得特别,但实际上增加了程序员的学习难度 (程序员大都掌握不止一种语言)。

· 传播困难。ABC 编译器很大,必须被保存在磁带 (tape) 上。当时 Guido 在访问的时候,就必须有一个大磁带来给别人安装 ABC 编译器。这样, ABC 语言就很难快速传播。

1989 年, 为了打发圣诞节假期, Guido 开始写 Python 语言的编译 / 解释器。Python 来自 Guido 所挚爱的电视剧 Monty Python's Flying Circus 。■本文未完,完整部分请浏览

http://developer.51cto.com/art/201302/380386.htm

为何MongoDB是NoSQL王者

10gen 公司 CEO Dwight Merriman 详细阐述了 MongoDB——第一款热门文档类数据库的诞生，并对动态 NoSQL 的未来趋势做出展望。他指出，Cassandra、Couchbase 和 HBase 将是 MongoDB 三大竞争对手。

时至今日，我们已经不可能把 NoSQL 的发展趋势与 10gen 公司的 MongoDB 割裂开来。当然，NoSQL 数据库家族可谓百花齐放，Andrew Oliver 还在他的经典文章《我到底该用哪种数据库》中进行了系统整理。众所周知，MongoDB 与其它同类开源竞争对手 Couchbase 及 Cassandra 的具体市场占有率也很难说得清楚。然而相信没有人会否认，MongoDB 已经成为新一代开发者眼中的至宝。在发展过程中，它很早就将简化 Web 应用创建与数据库扩展能力放在第一位，也正是凭借这一特色 MongoDB 才得以从英杰辈出的传统关系类数据库中杀出个黎明。

10gen 公司 CEO Dwight Merriman 于 2007 年与伙伴合作创办了这家企业——就在不久之前他才刚刚把自己建立的第一家公司 DoubleClick 卖给谷歌，交易金额达到 31 亿美元。Merriman 拥有令人艳羡的强悍技术背景：1995 年至 2005 年任 DoubleClick 公司 CTO，主持设计了 DART（即动态、广告、报告及定位）初始技术——这项技术如今已经成为网络广告投放的主要基础。据他自己介绍，当初启动 MongoDB 开源项目的原因在于“我们感觉有必要推出一种新的数据技术类型，而当时也正好是最合适的时间点。”在 10gen 公司的创建过程中，MongoDB 逐渐转型为商业开源项目，能够为用户提供包括技术支持及特殊认购版本在内的多层次产品供应。

为了进一步了解 MongoDB 的成长轨迹与影响力，Doug Dineley 和我上周一同采访了 Merriman。我们开门见山，从开创文档类数据库的灵感开始向 Merriman 展开了探索之旅。

问：您是怎样在经营 DobleClick 公司的过程中意识到市场对新型数据库的需求？

Dwight Merriman：之前的经验确实很有帮助。从某种角度上看，MongoDB 正是我希望自己能拥有的数据库产品。当时我们需要自来遍布世界的十二数据中心以及其中成千上万台服务器，还得确保每台设备都处于正常运行状态，这可相当困难。当然，上世纪九十年代的计算机在运算速度上完全无法与如今相提并论，这要感谢摩尔定律的一贯正确。在工作中我们需要一遍又一遍地对原有方案进行重新开发，这种枯燥而毫无价值的任务让我们希望创造出一种适合现代化软件、数据且能够与新时代工作规模相吻合的处理方式——同时还要与公共及私有云结构顺利接轨。但那时候市场上实在没有这种产品，于是大约五年前我们建立了 10gen 公司，并从零开始打造 MongoDB。

问：要说传统数据库技术的两大主要瓶颈，

为何 MongoDB 是 NoSQL 王者 II

可能关系类数据库的扩展性与控制性缺失最令人头痛。这个问题您怎么看？这两点是困扰您的主要原因吗？

Dwight Merriman :我赞同扩展位列其中，另一个嘛……我认为是数据模型。大家知道，关系类数据库可能是软件历史中出现过的最成功的技术，很多人都没意识到我们今天还在使用的技术已经有三十到四十岁的高龄。但如果将目光转向软件开发的另一大重点——编程语言，我们会惊奇地发现语言的更迭周期明显要快得多。软件开发方法本身已经发生改变，我们不再遵循瀑布式开发原则。如今，大家更青睐敏捷开发与多版本机制。编程语言不再局限于面向对象这一种，很多新语言都专为云计算而生，这种高度灵活性非常值得肯定。我们认为这种趋势也将给数据层带来重大变革，其影响之深远在过去 25 年的数据库发展历史中无人能出其右。

问：您建立了第一款获得广泛成功的 NoSQL 文档类数据库。这样的成就源自哪些重要积累？

Dwight Merriman :其实没什么特别的积累；我觉得整个职业生涯教给我的宝贵经验就是哪些可行而哪些不可行。我们研究过云计算——因为我们要根据扩展性需求决定如何编写代码——但却找不到理想中的工具。核心难题主要分两个方面，其中之一在于 :我们该如何向外扩展？扩展技术在理念上存在几道难以攻克的障碍，分配连接就是这样一道门槛。如果要向容纳有一千台服务器的集群进行负载分配，其难度可想而知。

针对这个问题，我们的态度是 :好吧，既然没有明确的解决思路，那就不要再为此浪费时间了。相反，我们尝试选择其它数据模型，希望在达到理想效果的同时回避上述难题。面向文档的数据模型由此诞生，从开发者的角度看我们对这一成果非常满意。尽管不太谦虚，但我们真的认为这套方案非常适合现在的代码开发者。

这有点像一种催化剂，指引着我们向正确的方向做出努力。我认为如今 NoSQL 家族中大家所看到的各类次世代、可扩展的非关系类数据库基本都具备这些特性。我真的很喜欢面向文档这一概念，因为它非常符合我们编写代码的方式。文档类数据库令信息变得可读，尤其是在开发者及数据库管理员眼中。在开发 MongoDB 的过程中，最令人印象深刻的点子就是把数据从代码中分离出来。我们应该有能力在不损坏程序的前提下查看数据库内容。关系类数据库在这方面表现不错，而我们应该继承这一优势。

作为文档的基础，我对 JSON（即 JavaScript 对象表示法）也很喜爱。JSON 赋予我们一套用于处理对象类数据的标准化独立语言，而且对于人类而言，它的可读性要比 XML 更高。据我所知有几款 NoSQL 产品属于 JSON 风格的面向文档型数据库。我认为作为这一领域的标准化数据模型范例，JSON 给大家带来了非常理想的切入点。

问：为什么要让 MongoDB 走开源道路？您是不是在项目启动之初就构思好商业模式？

Dwight Merriman :这么做也是出于多种因素的考量。首先，作为开发者我们很喜欢开源的概念，甚至可以被称为开源粉丝。当然，我们在坚持开源路线的同时，也相信自己有能力在开源这一领域创造出不断发展但又极具商业价值的免费项

目。红帽公司是一家了不起的企业，规模也相当之大，所以开源并不会成为业务道路上的绊脚石。

问：您的第一位客户是谁？

Dwight Merriman：我们的第一批客户来自 Web 2.0 与 startup world 网站。这要回溯到 2009 年，那时候 Shutterfly、Craigslist 及 Foursquare 等企业开始尝试 MongoDB。而一年之后，我们的产品迎来了更多规模更大的客户，例如 Intellisponse、O2、迪士尼乃至 eBay。时至今日，即使在企业层面我们也早已跨过了早期试用这个阶段。2012 年，MongoDB 的普及趋势开始向金融服务业界扩展，目前银行及其它金融机构开始广泛采用 MongDB 及 NoSQL 类产品——至少会在新项目中采用。甚至有不少企业表示：这是我们开发应用的默认方式。

问：他们指的是 Web 应用，对吗？

Dwight Merriman：哦，不是，因为 MongoDB 中不包含任何针对 Web 应用的特殊设计。从概念上说，这是一款通用型数据库。虽然刚刚接触 MongoDB 的客户会通过 Web 应用进行试水，但我认为人们应该把它推广到各个方面：内容管理系统、个性化系统以及移动流媒体等等。MongoDB 被用在 Web 开发中，但也同样被用于账目管理及离线分析等方面，这些都是历史数据存储量巨大的领域。总而言之，MongoDB 的适用范围非常广阔。

Dwight Merriman：客户所开发的企业应用之间存在共通点吗？因为在您的介绍中，我们会有种缺乏特色的杂烩感。

Dwight Merriman：使用范围的确很广很杂，而且你的问题比较宽泛，有点像：甲骨文关系类数据库包含哪些使用实例？我当然能够回答，但这既复杂又耗时。文档类数据库在数据对象符合面向文档的数据模型时表现出色，无论是从编程语言、分类效果还是数据合集角度看都是如此。

举例来说，某家电信公司自主开发了一套产品目录应用。由于公司规模极为庞大，产品数量也达到上万种，其中包括电话、延保产品以及其它一些服务计划。每种产品都拥有独特的属性，而由于 MongoDB 的数据模型非常适合这种工作，企业开发者会发现整个工作流程都变得更加简便。这算是个很好的使用实例。

我认为 MongoDB 的出色之处还不仅如此。内容管理应用后端、使用量巨大的移动应用、需要实时进行操作数据存储的在线应用等任务每秒钟需要处理数万次读写操作，而这也是文档类数据库的长项。

当然 MongoDB 也存在局限性，复杂度极高的工作或者对 SQL 及 UVC 的遗留请求进行处理就不太适合它。

问：您觉得自己的主要竞争对手是谁？

Dwight Merriman：Cassandra、Couchbase 和 HBase 是我首先能想象到的 MongoDB 三大竞争对手。

问：你对甲骨文 NoSQL 有什么看法？

Dwight Merriman：没有什么看法。

问：你会聘用班上成绩优异的学生吗？

Dwight Merriman：当然，我们很乐意这么做。

本文未完，更多内容请访问原文：

http://database.51cto.com/art/201301/378731.htm

■

不要为复用而设计

本文的作者 Elliotte 是一位著名的 IT 方面的作家，写 20 多本关于编程方面的书籍，有很多书籍在国内都有出版，其中包括《重构 HTML: 改善 Web 应用的设计》，《Java I/O》，《Java 网络编程》，《Xml Bible》和《XML in a Nutshell》等，目前他正在研究 XML 处理器 XOM、jaxen XPath 引擎和 Amateur 媒体播放器。

上周，一位同事的一个观点让我深受启发，这个观点是如此的显而易见，以至于当他说出来时我惊奇于为什么以前没有意识到这点：

如果你为复用而设计，那你就做错了。

你现在要写的代码的唯一目的就是服务于你目前手头上的需要解决的任务。不要为复用而设计。不要去考虑复用。不要为让代码复用而浪费一秒钟时间。

事实上，任何你需要的可以复用的代码都已经存在了。想要去连接一个 HTTP Server，并且要全面支持安全认证和 cookies 吗？这个东西听起来很多项目都可以用到，于是，你想把这个东西封装一下做成一个易用的 HTTP 类或共享包，很好的想法不是？

错。你应该使用 Apache HttpClient。

需要解决你的抛物线方法的初始值问题吗？不要去翻看你的《numerical analysis textbook》，你需要做的是下载 Flanagan 的 Java 科学计算库，或购买一个 NAG 许可证。想要给你的同事们做一个日历选择组件吗？直接告诉他们去用 JCalendar。尽管它在外观和使用方法上和你想象的不是完全一致，但完全够用。如果你打算做出自己的组件，或找一个现有的修改一下，你会发现，你开发出的这种不一样的表现效果并不适合另外一些人的应用，所以，不要浪费时间去开发自己的可复用的代码。

这些例子都是针对 Java 来说的，但对于另外一些主流的语言，比如 Perl，Python，Ruby，C++，C#，Scala 等，都是适用的。

事实上，如果一种语言不能提供解决你的问题的可复用的代码，那你就是选错了解决你的问题的语言。

有例外的情况吗？我只能想出两种（目前为止我感觉没有第三种情况了）。

第一种例外是你在开发一种新的东西，你需要的类库不存在，你是第一个进入这个领域的人，你需要写出可复用的代码。例如，当我率先开发出 XIncluder 类库时，XInclude 的规范还处于制订中，你在 Java 里找不到第二个可用的类库。我写的这个类库成了规范的可实现的一种证明，推动了规范向更完备的状态发展。十年前开发我自己的 XInclude 类库是明智的，而今天绝对不会再重做这样的事。

第二种例外情况是针对专家的，我甚至还不确定这是否是例外。如果你是某一个领域的真正专家，有可复用的代码能解决你的领域的问题，而你经过认真的研 究现有的解决方案，你认为它们是不完善的。■本文未完，完整部分请浏览

http://developer.51cto.com/art/201301/379024.htm

程序员，都去写一写前端代码吧

前端开发，则是这“多方面的事情”中的一个重要方面。潜心尝试过的人兴许会有这样的体会，这是一片崭新的世界，无论是理念、技巧，都有一种新鲜的感觉。

你可以认为我是一个极端的人，就像有许多人专注于自己的领域而不屑于其它“肤浅”的工作范畴一样。比如我见过不少认为做 portal 没有技术含量的判定，做工程都是充满苦逼行为的言论，最近则还有那些“大数据”崇拜者的疯狂吐槽……我的极端则有些不同，我的极端在于我认为绝大多数优秀的程序员，都要尝试多方面的事情。并不只有底层开发或者机器学习充满睿智的挑战，我做了几年网站，很难说这就是我最初的兴趣，虽然也在接触和学习其他的领域，但是依然觉得，做网站仍旧充满挑战，互联网真是一个奇葩充斥的地方

前端开发，则是这“多方面的事情”中的一个重要方面。潜心尝试过的人兴许会有这样的体会，这是一片崭新的世界，无论是理念、技巧，都有一种新鲜的感觉。如果你还没有尝试过，相信我，它会丰富你的视野，至少在设计和编码上，你会有崭新的认识。

JavaScript 代码是存在诸多天生缺陷的，你可以找得到太多它的替代品和改进品。另一方面，它确实给了程序员很少的限制——如果你写过 perl 代码，你大概也深有体会，什么样的代码是自由的代码，什么样的代码是充满诗意的代码。与之相对的大概是语法严格的 Java 代码，就像老实、规矩的孩子，他不会带给你多少破坏性，但是也没法带给你丰盈的代码美感。但是 JavaScript 有 N 多类库，有足够活跃的语法自由度，有 eval 和 prototype，还有那些动态语言的特性，你可以写出许多飘逸的代码。

另一方面，代码的自由一定带来代码层面规划和解耦的艺术。如果代码还处在漫山遍野全局 var 和全局 function 的温饱阶段，那么肯定是无法感受到这一点的，而且在这个阶段也根本称不上会写 JavaScript 代码。有许多人说前端开发简单，如果只是把它理解成为“好上手”，或者说 alert 一个字符串，改变一个 div 的颜色，那它还真是太好学了。再加上 CSS 的方便和简陋性让它连编程语言都算不上，而 HTML 又是容错性非常强的标记语言，所以你可以很容易写出能看到效果的界面来。

写一个 UI 稍微复杂一点的产品代码，就会无比地感受到规划和解耦的力量。无论是 HTML、CSS 还是 JavaScript，变量或者对象都是极易被污染的，“模块化”显得举足轻重。在 Java 的世界里，你的武器很少，包、类、加载器又在你无意识的时候把这些繁琐的模块化的工作轻易化简完成了。但是写前端代码的时候你发现需要自己去考虑了，比如页面的分块布局、CSS 的继承树、JavaScript 的绑定和匿名函数，还有那么多开源的库来帮助完成模块化。

■本文未完，完整部分请浏览
http://developer.51cto.com/art/201301/378010.htm

回归结对编程

简介

我这一生中，很少进行结对编程，当然非正式的除外（例如和别人一起解决 bug）。在 Typemock 工作的时候，我大部分时间都在改 bug。

当被问及提高代码质量的最好方法时，我一般回答是：代码审查。代码审查是最佳的减少 bug 的方法。但我更喜欢的是 —— 结对编程。

我这一生中，很少进行结对编程，当然非正式的除外（例如和别人一起解决 bug）。

在 Typemock 工作的时候，我大部分时间都在改 bug，没有特定的做法，没有人指导应该怎么合作，仅仅随意地和不同的人解决不同的问题。

最近我又接触到了结对编程，我现在的感觉和以前完全不同。我更加注重这个过程，而以前的我仅仅对结果感兴趣。

以下是我最近总结的一些要点。大部是作为专家的拍档的角度来说，有些也从主导人的角度。

氛围．开始一段工作的时候，你大概可以估摸到事情会怎样发展。你有多投入，你的拍档有多投入，对问题和代码有多熟悉，这些都很重要。你们最后能做到多好，决定于你们的开始。

耐性．这是说给急性子听的。我不知道以前自己是不是有耐心，现在我需要下大功夫去培养一下耐心。当我是主导人的时候，我常常需要不厌其烦地对拍档作出巨细无遗的解释。

静心聆听，不要一味反驳．当我不是主导人的时候，我会一次又一次地要求主导人解释他的方案是怎样解决问题的。当然，我需要安静地听他讲解，才能听明白。但是，要我一声不吭的，太难了。

不当沉默的羔羊．当我不是主导人的时候，我常常在心里衡量，要不要提出我的疑问。说出来的话，怕他会笑我笨。沉默的话，的确不会引来嘲笑。但是我还是常常逼着自己提问，一般这样开头：“可能是我太笨，但是我不太明白……”。结果，有时候我们会就这个问题讨论下去，有时候拍档则会像看傻瓜一样看着我。

态度很重要．我自己的态度固然重要，但是拍档的态度也很大程度上影响了结果。例如，当我不是主导人的时候，对方有些言语和举动使我感觉到自己好像拖慢了进度。我经常告诉自己不要那样想。当我是主导人的时候，我尽量控制自己的言行举止，以免拍档误会。当主导人与不，都难啊。

拍档也是很重要的．无论是菜鸟还是高手，都没有完美的拍档，只有最适合的拍档。有时，你不能选择和谁合作，但你拍档总有值得你学习的地方。

学会协商．即使是这个层面的合作，也脱离不了付出和收获的关系。你需要根据进度决定什么时候讨论相应的话题，从命名到测试点。

你的拍档可能同意，或者不同意，又或者与你击掌欢呼。失败总是存在的，做好心理准备吧。

■本文未完，完整部分请浏览
http://developer.51cto.com/art/201301/379129.htm

进阶过程：程序员做项目的独立性

第一阶段 :编码机器

这是最低级的阶段,程序员拿到详细设计文档,上面连许多方法接口都定义好了。重构一些代码,写一些实现,调用一些既定的 API,然后花许多时间在各 种各样的场景测试上面。从做的工作上看,这都不能算程序员,最多,只是编码技巧卓越的码农而已。它几乎扼杀了一切创造力,这很常见,比如在一些对日外包公司,就是如此。

第二阶段 :独立的实现者

程序员得到的只是粗略的设计文档,也许注明了外部接口的清单,还有框架和基础设施的 API,需求已经澄清清楚,接下去要做的就是发挥聪明才智把软件 实现设计好,把代码写好,测试通过。这项工作可以在安静和独立的环境中完成,因为没有什么是不够明确的,那些本不清楚的部分,项目经理、架构师和产品经理已经搞定了。这样的环境下可以诞生许许多多 OO 设计优秀、代码清晰简洁的程序员,但是这始终只是在做一个残缺的项目而已。很多程序员新手都是从这个阶段开 始的,但是程序员如果只会代码层面的设计、编码和测试,并不能算一个完整的程序员,也许连一半都算不上。

第三阶段 :项目沟通者和管控者

程序员要和需求工程师,甚至客户澄清需求,分析可行性 ;需要自行分析设计项目中的技术难点 ;参与决定和管理迭代周期和计划表 ;组织和参与项目组内运 作跟踪会议。

编码以外的事情会占用一些时间,这些时间大多来自沟通的成本。如果说,前两个阶段还未能明显看出沟通的重要性的话,那么到这个阶段,一个不能 良好沟通的程序员,将成为项目组运转的瓶颈。国内至少有一半的软件公司的程序员做项目应该处于这个阶段。

第四阶段 :从做项目到做产品

从做项目跨越到做产品,想想容易,做起来很困难。做项目需要更多倾听用户需求,但是做产品更注重思考,思考用户的痛点和产品的定位远重于倾听用户表 述。

从项目周期上看,做项目关注在拿到需求并实现交付的过程,但是做产品需要把更多精力花在产品定义、设计,还有长时间的产品维护上。

做产品的程序员,必 须要和客户沟通,必须要维护自己开发的软件,获知用户和市场的反馈,如此才能体会到什么功能才是迫切需要的,该怎样把技术、业务和实际产品的实现结合起来。

第五阶段 :产品成长的见证人

也许很少人能够参与从零开始,经过创意、市场分析到产品设计的过程,在明确要做什么之前有大量的时间会花在产品探索性的工作上面。也许会做很多的产品原型,也许某些版本和功能在 A/B 测试之后就被放弃了,更有些产品在流传开来以前就销声匿迹了,或者很快就死在抄袭和山寨手里。

产品的更迭和换代总是千辛万苦,而看得到的部分往往如此简单,但是谁知道它的历史有多曲折呢? ■

■ 编者按

网易有道笔记负责人谈敏捷开发的实战经验 :什么时候适合使用“敏捷开发”呢？我们的经验是需要两点 :一、团队有三名或以上的研发工程师 ;二、团队内有一名合适的 Scrum Master。

蒋炜航：敏捷开发的实战经验

网易有道笔记负责人谈敏捷开发的实战经验 :什么时候适合使用“敏捷开发”呢？我们的经验是需要两点 :一、团队有三名或以上的研发工程师 ;二、团队内有一名合适的 Scrum Master。

作者 :蒋炜航，网易有道笔记负责人

注 :名词详细解释见文末

有道云笔记团队成立于从 2010 年，从成立伊始我们就一直积极地在实践中尝试 Scrum(敏捷开发的一种项目管理方法) 的做法。到 2012 年底，3.0 发布时，我们在 5 个主要平台 (PC、iPhone、Android、iPad、Web) 上总共发布了 46 个版本，累计了近千万激活用户。在这个过程中，我们逐渐摸索出一套适合以产品和技术创新为核心的中等规模 (数十人) 研发团队的 Scrum 实践经验。

1、Scrum 不是万能药，要在时机成熟时推行。

什么时候算时机成熟呢？我们的经验是需要两点 :一、团队有三名或以上的研发工程师 ;二、团队内有一名合适的 Scrum Master。

刚开始的时候，一个开发团队可能只有一名或者两名研发工程师。这时候并没有全面推行 Scrum 的必要，而可以借鉴 Scrum 中的一些做法。比如有道云笔记的 Web 团队最初就是这个情况。当 Web 团队只有一名研发工程师时，我们就尽可能地尊重他的工作方式。同时为了保证项目进度可控，我们引入了 Scrum 的 sprint 机制——以 sprint 为开发周期，每个 sprint 进行一次 Web 产品演示。这不但能够让工程师有一个以 sprint 为期限的压力，还能够让其他同事即时地了解项目的进展，以便做出相应调整。当 Web 团队扩充为两名工程师时，我们又引入了结对编程、持续集成、相互代码审核等做法。直到 Web 团队的规模进一步扩张时，我们才开始考虑全面启用 Scrum。

当团队内无法找到合适的 Scrum Master 时，不要轻易推行敏捷。如果你的团队是由新人组成，或者即使有资深员工但是他并不了解或认同敏捷开发的话，那么你需要等待合适的 Scrum Master 出现。

合适的 Scrum Master 需要具备几个特质 :首先，他要认可敏捷开发这种方式 ;其次，他要熟悉业务，起到教练的作用，能带领团队走正确的流

程 ;并且,当团队遇到问题时,他要有能力和担当引导团队做出决定,在团队成员遇到困难时,他要协助成员解决 ;最后,他要能识别重要和紧急的事情,而并不是事无巨细的反馈到产品负责人那里。

敏捷开发虽然希望团队自我管理,但是这需要一个过程,开始的时候,一个合适的 Scrum Master 至关重要。有道云笔记的 Web 团队在成立一年多以后才开始推行 Scrum,很大的一个原因是在培养合适的 Scrum Master。依据我们的经验,最胜任 Scrum Master 的人选是技术主管。我们也曾尝试过让产品经理担任 Scrum Master,但是由于产品经理本身往往担当产品负责人,兼任 Scrum Master 会影响他在产品机会和产品体验等方面的投入。

2、限制 Scrum 团队的规模,建立 Scrum 团队之间的协作机制。

随着业务的发展,团队会变大。这个时候不拆分团队的话,效率会变低。

有道云笔记移动端团队就经历过这样一个过程。很长一段时间 Android 和 iOS 的研发工程师组成一个 Scrum 团队,有共同的产品负责人和 Scrum Master。但是随着移动端团队人数的增长, Scrum 会议的效率却降低了。虽然 Scrum 会议只有不到半小时,但是当说一个平台的事情的时候,另一个平台的工程师会觉得无所事事。发现了这个情况后,我们把移动端团队按照平台拆分成了两个 Scrum 团队,以确保 Scrum 会议上说的是每一名参与者都关心的事情。总的来说,参加 Scrum 会议的所有人,包括产品、开发和测试,不应该超过 9 个人。

按照平台拆分团队,限制了 Scrum 团队的规模,提高了 Scrum 的效率。与此同时,多个 Scrum 团队之间必须进行有效的协作。

在初期,我们鼓励研发工程师通过面对面地商量,快速推进来处理平台之间协作的需求。但是随着业务的发展,这样的协作越来越多,也越来越复杂,这样面对面的讨论往往会疏忽细节需求。比如说,有道云笔记 3.0 版本中的待办事项功能,就需要 PC、Web、Android、iPhone 以及 Server 等多个 Scrum 团队一起,对这个功能进行产品定义和确定技术方案。这样复杂的协同需求需要额外的机制来保证。这个机制就是 Scrum Master 的定期会议。在这个会议上,我们会讨论各个 Scrum 团队相互依赖的项目,安排好各 Scrum 团队的开发顺序。对某一件具体的事情,其中的一位 Scrum Master 会被指定为具体负责人来驱动跨 Scrum 团队的协作。同样,只有当 Scrum 团队间的协作任务比较复杂的时候才需要引入这个机制。

3、产品经理和研发工程师要拥抱 Scrum 带来的变化。

在引入 Scrum 之前,一般的项目管理方式是版本式 (瀑布式) 的,产品经理决定下一个版本做什么,预期发布的时间,然后由产品负责人或者技术负责人来兼做项目经理。这个时候遇到的问题是项目往往会延期,但是产品经理会有一种对项目把控的感觉。

引入敏捷开发之后,这个事情变了,发布是跟着 sprint 走的。基于持续交付的原则,一次发

蒋炜航 :敏捷开发的实战经验 III

布包含一个或者多个 sprint 的内容,而这些内容是由团队整体决定的,而不是产品经理个人决定。产品经理只是定义了功能需求的优先级,这些功能需求与代码重构、开发工具、以及市场运营等的推广支持等需求一起排期,最后由整个团队决定一个 sprint 做哪些东西。

从表面上看起来,产品经理对产品的把控小了,为此,团队一位资深产品经理有过质疑。最后,我们还是说服了他接受敏捷。事实上,接受 Scrum 并不困难。这样,产品经理可以把重心放在对产品需求的把握上,而不必整天问这个咋样了那个咋样了。而且,团队的开发效率,功能点完成的速度并没有因此而降低。

研发工程师同样要拥抱 Scrum,调整自己的工作方式。Scrum 不鼓励过度设计,而采用涌现式设计。

这意味着开始往往不会把技术架构做得大而全,而是鼓励快速出成果。当然这并不是说程序架构能够设计得很糟糕,而是说不要花太多的精力在未知的事情上,小步快跑。为此,代码重构是必须的 (编者注 :在不改变软件现有功能的基础上,通过调整程序代码改善软件)。我们并不建议整个 sprint 都去做重构,更建议持续重构,把代码调整也分解成任务,每个 sprint 做一些。在一些大版本发布之后,重构任务的比例可以适当高一些。

4、量化衡量团队的执行力的指标 :完成度、评估准确度、计划合理度。

当 Scrum 团队不大的时候,可以依靠主观感觉来评估执行力。有道云笔记团队在初创的一年内,对 sprint 的完成情况是没有量化的评估的。

Scrum 教材对执行力的量化评估用的是故事点和速率。

由于团队成员实际上都是有道云笔记的用户,能够直观理解产品经理提出的需求。因此我们省去了用户故事,由产品经理提产品需求,而团队把需求分解成任务。经过一段时间的探索,我们定义了几个量化指标,其中最重要的是完成度,我们用这个指标来衡量团队的执行力 :

完成度 =1- 计划内未完成任务的剩余时间 / 计划内任务评估时间

(完成度的数值在 80~90% 之间比较好。过高的完成度说明 sprint 计划过于保守。)

有些管理者会怀疑完成度的准确性 :第一是,团队是否会把计划内任务的评估时间评估得过长,使得完成度看起来高? 事实上根据我们的经验,团队对计划内任务的评估往往是偏乐观的 ;第二是,计划内未完成任务的剩余时间是如何出来的? 这个是由团队在 sprint 末尾评估出来的,因为这时技术设计多已完成,这个时间是比较准确的。我们也曾尝试过燃尽图,发现并不如完成度来得直观。

另外,我们还定义了两个指标来作为辅助参考。一个是评估准确度 (计划内任务评估时间 / 实际使用时间),一个是计划合理度 (计划内任务使用时间 / 计划外任务使用时间)。这两个指标的历史数值可以让我们更加了解团队执行的情况。■

原文未完,请参考

http://developer.51cto.com/art/201301/378700.htm

影响项目的因素及经验总结

项目管理是一门学问，需要一定时间的工作经验积累，并不是去考个 PMP 回来就能当项目经理的，真正注重的还是实践，照搬理论那一套的话，很多公司内部的现状都不足以支撑 PMP 所要求的完善体系，大型公司可以试下。

最近做了很多从无到有的产品项目，有成功的也有不完全按期完成的，甚至有两个项目已经因为开发资源问题超期了 1 个月。成功项目的经验可以借鉴，为后续项目铺路，失败的项目则可以当做经验教训。我们都要学会从项目失败中吸取教训，只要我们能够勇敢的去面对它，那么犯错也不见是一件坏事。

很多公司的产品经理其实都还承担了项目经理的角色，管理项目也是产品经理的必备能力之一。

导致影响项目失败的因素其实不多，很多时候我们无法预先避免这些因素的发生，那么我们要做的就是如何去弥补。个人总结了一部分，也从网上资料学习到了一些，会知道项目失败的因素主要分为以下几类：

技术原因：

1、领先技术的诱惑，对新潮技术的尝鲜，但技术基础储备不足；

2、不完善的技术设计，基础技术架构没有搭建好，或者是设计不完善；

3、为非技术问题提供了技术解决方案，没有认真思考问题的本质；

4、依赖软件包（JDK）来满足需求，过分依赖会影响扩展性；

5、在开发生命周期过程没有充分利用工具；

6、以技术为导向进行开发，不是需求导向；

人为因素：

1、缺少行政人员的支持，一般是指项目相关的部门，如服务器采购需要采购部配合，也有时候是指主导项目发起的领导；

2、缺少领导，团队内部多头领导；

3、没有敬业精神的项目团队；

4、功能不全的项目团队，人员配比缺失；

5、管理第三方的因素，如技术外包的项目；

6、缺少一个项目精英，如专职的项目经理；

7、缺少项目所有权；

8、相关人员冲突，指人员配比失衡；

9、拒绝变更，无法面对需求的变化；

10、不友好的组织文化；

11、经验不足的项目经理；

12、缺少商业理由，项目的价值较低；

13、不清晰或模棱两可的优先级；

14、缺少培训，项目执行过程中状况百出；

15、相关人员动机不一致，缺乏向心力；

过程管理因素：

1、缺少项目管理方法体系；

2、缺少系统开发方法体系；

3、缺少收益管理方法体系；

4、缺少质量管理方法体系；

5、未能确定和转移项目风险；

影响项目的因素及经验总结 II

6、未能管理好需求；

7、过长的项目时间表；

8、测试覆盖不足；

9、计算机化的” 爆炸 “方法

从失败中吸取教训是不断改进过程的重要组成部分，下面罗列一些主要的经验教训。

管理用户预期

即项目人员要从一开始就明白需要交付什么以及不要交付什么，要在项目中确定用户的需求和建立尽可能清晰的所有权。即使在最好的情况下，用户以前收到的信息也是有限的。通常情况下，我们很难确定能够提供反馈信息的合适用户。在项目一开始就需要确定主要的用户需求，并且为主要用户提供时间，以便他们确定所有的需求，同时他们也有责任提供和验证信息并投入相应的资源。

项目规格说明书必须考虑价值和用户需求

第一，项目是因为可确定且可测量的用户需求而产生并发展的。在项目初期确定的清晰目标将随着项目的进展而逐渐变得模糊，这是交付期限过长的项目所共有的特点。因此在项目开始之前，需要确定最终用户，以便在项目的设计和开发过程中充分考虑到他们的需求，同时用户也有责任而且需要采取相应的行动来帮助项目获得成功，这一点非常重要。用户需求构成了项目分析和设计阶段中一个至关重要的环节。需求确定后，就要为这些需求确定基线，并将它们引入到项目管理系统中，同时使用变更控制对其进行管理。如果这些需求出现了变更或添加了新需求，则需要对项目进行影响分析，并对项目计划进行相应的修正。

第二，项目规格说明书必须关注项目价值而不是技术解决方案。因此，即使从技术上讲已经存在明确的解决方案，但在进行项目评审时仍需将重点放在与项目价值相关的方面。

在确定资源前测量和评估项目的规模和复杂度（重视实现性）

技术力量的发展带来了一个不好的后果，就是让我们相信，许多以前不可能实现的目标如今不但可以实现了，而且可以轻而易举地实现。

有时候这种想法在项目的早期阶段通常表现为对项目的潜在收益过分夸大、过于庞大的项目范围定义以及过分乐观却相当危险且不够详细的项目规划。

项目的规模和复杂度是项目成功与否的一个决定性因素，因此我们需要明确确定的是：

a、提议的项目进度表是否现实可行；

b、项目的需求案例是否可行；

c、解决方案在技术上是否可行；

新技术的引入必须安排相应的培训

新技术或者一些开源工具会给项目带来的不少影响，容易导致有关程序员角色和责任的不明确。因此在项目计划中纳入培训成本和时间进度以确保员工知道如何使用和维护系统是至关重要的。没有合理的培训就是永远不可能实现软件投资的全部潜在收益。更重要的是，缺少培训可能会为项目带来实现风险和运作风险，这些风险可能会最终威胁项目的长期可用性。■

原文未完，请参考

http://developer.51cto.com/art/201301/377368.htm

编者按

我的同事朋友 Chris Eargle 写了一篇关于新年计划的有趣文章。他让我想到了，没有出现那场世界末日是我们多么大的幸运呀（还有其他我这 45 年中躲过的天灾），于是，我也有了一些我自己的以程序员为主题的新年计划。

程序员2013新年计划

我的同事朋友 Chris Eargle 写了一篇关于新年计划的有趣文章。他让我想到了，没有出现那场世界末日是我们多么大的幸运呀（还有其他我这 45 年中躲过的天灾），于是，我也有了一些我自己的以程序员为主题的新年计划。

找到一名导师 / 成为一名导师

在你的职业生涯中，你能做的会给你带来最多麻烦的事就是成为屋里最聪明的人。我说的并不是你坚信自己你就是屋里最聪明的人。我的意思是你成为团队里真正的万事通。问题终结者。终极疑难解答者。

于是，这就有了另外一个问题：你有疑问了去问谁呢？

如果你的回答是“谷歌”，那你是不思进取。去到那些你认识的（或不认识的）最聪明的人中间去。参加你们的本地社团。去你们本地的编程活动中发言，去和其他的讲演者一起喝酒聊天。找那些你可以接触到的人，让他们成为你的导师。

找到一名导师

我在生活中有好几位导师。他们是我尊敬的人和能让我轻松问问题的人。有些人甚至非常的专业！没错，这些是我软件开发圈外的导师。

如何去请教你的导师？这取决于你。我是在有问题时找他们。我对他们说喝杯咖啡吧，找个地方坐下来，聊聊天。如果我们能同一个城市的某个研讨会上遇到，我会和他们一起出去喝酒吃饭。早些年，我很注重形式礼节，特别是我作为团队的消防队员的时候。如今，我已经不再有任何形式拘束了。更多的是随心所欲的求教。

成为一名导师

同样，我们也应该成为生活中的某些人的导师。如果你有孩子，你已经承担起了一名导师，父母，朋友，老师的职责。当然，对于一些同龄人的指导，我们需要去掉父母那部分的角色。对他们你是不能发号施令的。

如何让自己成为其他人的导师？当然，如果有人来请教你，那是最好了。这就有些名正言顺了。但你也可以在不声明“我是你的导师”的情况下成为某人的导师。看看是否有人在为一些事情愁眉不展，你可否帮助他们？对他们说喝杯咖啡吧（如果是九零后就喝红牛）。去跟他们一起吃饭。跟他们聊天。更重要的是，倾听。指导并不是宣扬你的智慧或你的经验。导师是要成为一

个耳朵，一个肩膀，一个指点方向的手指——在他们需要的时候。

KISS

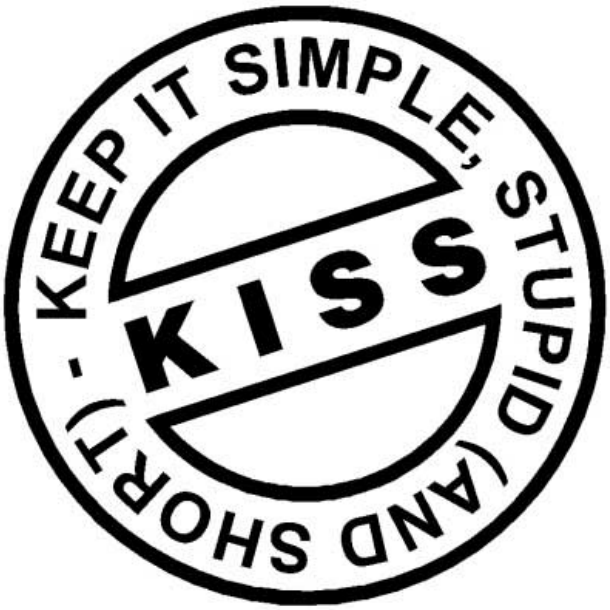
“贝丝，你在呼唤我，但我只是想去底特律摇滚城市里每天没日没夜的摇滚”！ [歌曲 KISS 的歌词]。似乎这个社会在召唤你制定一个加入这种 KISS 大军，去吃喝玩乐，去体验生活的新年计划。但我在这里说的可不是这些。

我说的是 Keep It Simple, Stupid！作为程序员，我们绝大部分时间都是花在了维护代码上，只有少部分的时间用来创造代码。

事实也证明，维护代码要比写新代码要难的多。所以，按照这种逻辑，如果我们在创造代码时极尽所能写出最巧妙的代码，那么我们就没有足够更高的能力来维护它们了。

Blaise Pascal 在他的第 16 封省府信件中说“这份信件很长，原因很简单，我没有时间使它更短”。抽出时间，重构你的代码，让它们更短小。让它们更容易阅读。尽所可能的在所有地方遵循 SOLID 原则。

如果你不能把它向一个 9 岁的小孩解释明白，这说明它太复杂了。公司雇你不是让你来表现脑瓜好使的，如果你写的代码没有任何人能接手维护，你不会因此而得到加薪或晋升。



去读该死的手册 (RTFM)

这是我的第一次圣诞节里不需要在平安夜里去做一些东西。在以前，我会做小脚踏车，布娃娃房子，布置厨房，以及所有类似的东西。当然，做这些东西都不需要参考手册，只是需要在孩子们上床睡觉后才能开始，而且第二天早上天蒙蒙亮就会被三个孩子跳上床来吵醒。噢，这些美好的回忆！

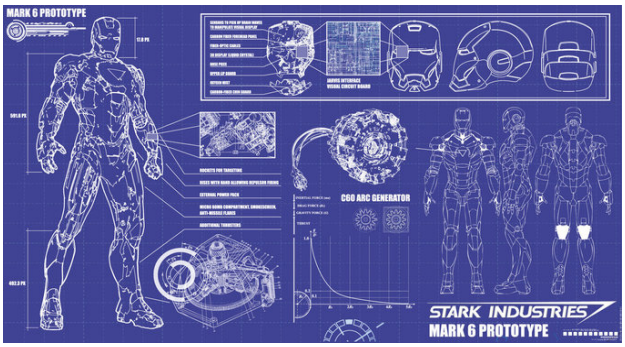
当然，所有的这些不眠之夜都有一个相同的主题。我知道我不需要读操作手册就能做这些。可工作中更常见的是，文档上的图表画的一团糟，文字是经过了三种不同语言翻译过来的，我对这些文档的质量的意见一致很大。■

原文未完，请参考

http://developer.51cto.com/art/201301/377779.htm

像建筑设计师一样去写程序

作者 / 果子



写代码和写文章，从某种程度上是相通的，需要逻辑、构架，也要尽可能的简练。

我们之前说过，创作者的时间表和管理者的时间表是不一样的，编代码和写文章，都是一个孤独而不能受到干扰的过程，面对屏幕，就是一场自己与自己的战争。

同样，如果说建筑师最后的成品是建筑的话，那么程序员和软件工程师最终的成品就是软件。在实际动工之前，建筑师将会将建筑的每一个细节，都在蓝图上加以呈现。只是程序员和软件工程师并不会这么做。或许，这就是为什么房子很少倒塌，而软件却经常崩溃的原因？

设计蓝图会帮助建筑师确定他们的设计是可行的。“可行”并非只是保证不倒塌，它同时还意味着，建筑能够达到预定的服务于人的目标。客户或者开发商，也是通过蓝图去了解一个设计师的想法和他正打算去做的事情。

相较之下，很少有程序员会在他们开始写代码之前，连一个粗略的框架都没有。

大部分的程序员都认为：所有不能直接产出代码的事情都是没有意义的。思考并不能直接转换成代码，但是倘若在没有一个整体框架之前，就开始匆忙码字，这也是没有意义的。在程序员开始写代码之前，他们应该先充分理解这些代码最终所要实现的功能。理解的过程，自然需要去思考，而将思考过程写下来，对于程序员来说又是件很耗时的事儿。

但漫画家 Dick Guindon 曾经所过：

写作是发现你想法有多糟糕最好的方式。

蓝图帮助我们理解建筑的构架，同样，在我们开始噼里啪啦写代码之前，我们也需要一个类似的“蓝图”，也就是“注释”(specification)。

“注释”不能直接产出代码，所以被很多程序员忽视。但没有“注释”，直接开写，就好像让建筑承包商没有设计师的图纸就直接上阵一样。

也有人会说，将程序员和建筑师类比，并不合理。因为拆墙重建困难，但删掉重写则相对容易，所以，程序员可以先写着，不满意再改。

这种想法是错的。为什么呢？因为 Debug 的过程也非常耗时。

我最近也完善了一些程序，这个过程需要对程序构架本身有个非常清晰的了解。我花了接近一天的时间去了解整个程序的运作机制，而倘若有注释的话，这可能只需要 5 分钟的时间。

为了避免引入 bug，我需要理解任何小的调整可能带来的结果。而没有注释，使得我必须花很长时间了解每条代码的含义和作用。尤其对于上千行的代码来说，首先读懂它就很耗时，想要改掉其中的某行，我必须了解小的调整可能对整体架构和前后逻辑造成的影响。最终，在超过一个

像建筑设计师一样去写程序 II

星期的时间里我只改了 180 行代码，而这对一个动辄千行的程序来说，改变算是很小的了。

Debug 只是写代码一个小的组成部分。

这数千行的代码很多我 10 年前写的，尽管我对它们仍有些许残存的记忆，但如果有注释的话，修改代码的过程会更加顺畅，我不仅能在最短的时间里读懂整个框架，还可以准确定位我所要修改的部分。

改别人的代码就更加困难了，每个人的思维方式都不一样，如果没有注释，我通常要花费两倍以上的时间，只是为了修改一些细小的错误。

那么，我所谓的“注释”又是指什么呢？“注释”是指附在代码之后的，一段形式化规范说明的文字。

但需要区分开的是，如果只是去打造一个工具室，我们是不需要一整套摩天大楼蓝图的，同样，对于小的算法来说，我们也不需要给每条代码加上注解。

我最近要编写的程序，最多称得上是“平房”，而非摩天大楼。

我会为我的每个算法附上注释，有些非常简单的算法，我通常只会插上一两句注解。我有一个非常简单的法则来帮助我和其他人了解我的程序 :注释应该尽可能有效地去帮助任何一个人理解和使用我的代码。

一旦我知道一行特定代码要做的事情后，写的过程其实是非常简单而直接的。

也有些程序需要用到非常规算法，这时我会写下我算法的主要思路，来试探其可行性，同时也帮助我更高效的 Debug。

除了那些特别重要的代码，通常我的注释都是非正式的。

在过去的十年里，需要我写准确并正式注释的次数并不多。但对于一个非常复杂的系统来说，注释的重要性不言自明。

很少有工程师会在构建一个复杂系统时，花时间去写好注释。有些学校也会教你怎么写注释，但更多时候都是教你如何写好代码。这需要实践，如果你没有画过搭建一座平房的图纸，你很难直接画出摩天大楼的蓝图。

写好注释也没有一个简单的准则，但有一点你要尽量避免，那就是用代码去解释代码。就好像你不能用两个人们都不理解的东西，用其中一个去解释另外一个。

建筑师也不能直接用砖块来告诉你，他想搭建一个什么样的房子。

了解一个复杂的系统，最好的办法就是将其核心用简单的概念，抽象的概括出来。初中数学的一些基本概念可以用来帮助你写好注释，比如你可以用一些集合、方程和简单的逻辑来解释你的代码。

对于一些复杂的算法，你也可以引入数学里没有过的概念来加以解释。总的来说，如果你的注释偏离一些抽象的数学基本概念越远，理解起来也就越困难。

思考并不能保证我们不犯错，但不思考，犯错是无法避免的。

注释能帮助我们将错误最小化。

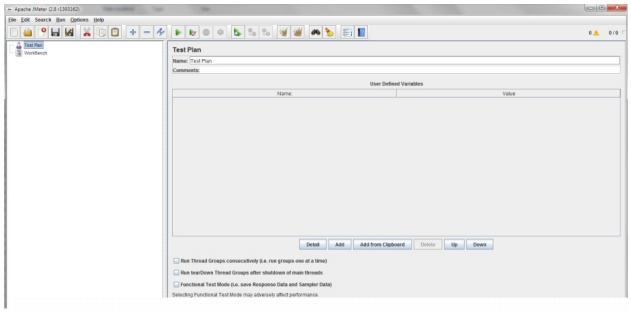
同时它还能提高我们改错的效率，节省我们的时间。■

使用JMeter进行HTTP负载测试

在本文中，我将介绍如果在 JMeter 中设置一个最基本的负载测试计划。一般来说，你想通过负载测试来获得以下两个问题的答案。

以前我写过一篇文章：high availability load balancing with HAProxy and CARP on FreeBSD，其中我使用了 JMeter 作为测试工具。在本文中，我将介绍一些如何用 JMeter 来进行 HTTP 负载测试的基础知识。JMeter 是一个 java 应用程序，用于模拟产生一个 高负载到一个服务器上，以测试这个服务器的承载强度，或用于分析在不同的负载下的服务器的性能。完整的 JMeter 情况，请到其官方网站上了解：这里。

你要明白的一件事是，这个工具并不是一个浏览器。这意味着 JMeter 并不会执行提供给浏览器执行的所有功能，它也无法运行 JavaScript 或 Flash。它只是一个能运行在 windows 上或其它操作系统上的一个桌面应用程序。因此，请首先到这里 把它下载下来。如果你使用的是 windows 操作系统，解压后你就可以运行“bin/jmeter.bat”命令来启动它。你将看到下面的显示：

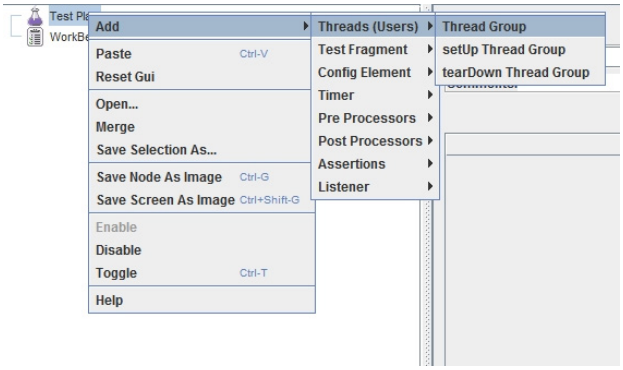


在本文中，我将介绍如果在 JMeter 中设置一个最基本的负载测试计划。一般来说，你想通过负载测试来获得以下两个问题的答案：

– 你的应用程序能够承载多大的用户量?

– 在什么样的负载下，你的应用会崩溃?

因此，一开始，首先你要添加一个线程组(用户数)：



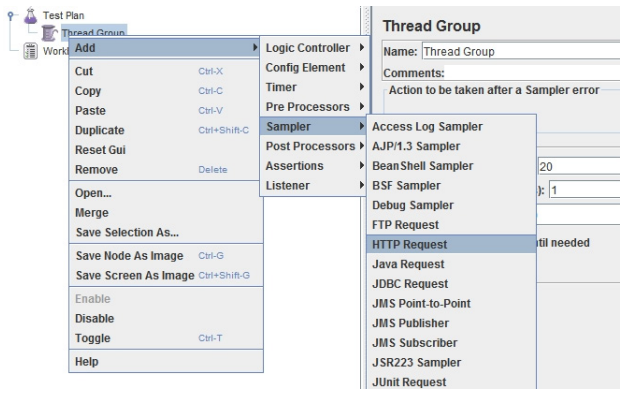
然后，开始设置这个线程组：

– 用户数

– 过渡期（用户组发出请求的间隔时间）

– 循环次数（这个线程的运行次数）

其次，你需要增加一个例程 (HTTP 请求) 到这个组中：

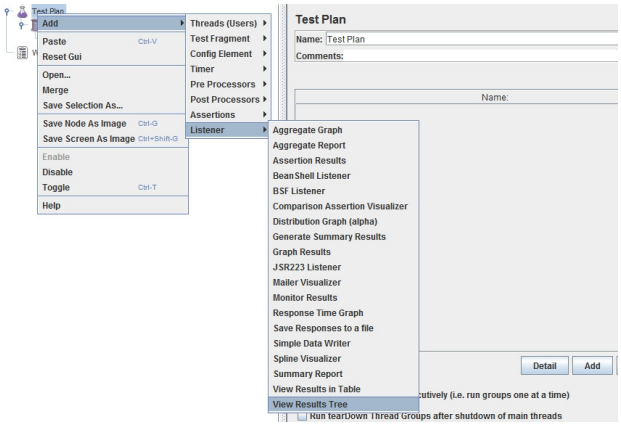


使用 JMeter 进行 HTTP 负载测试 II

一旦你增加了一个 HTTP 请求例程后，你将看到大量的选项。你需要注意的是：

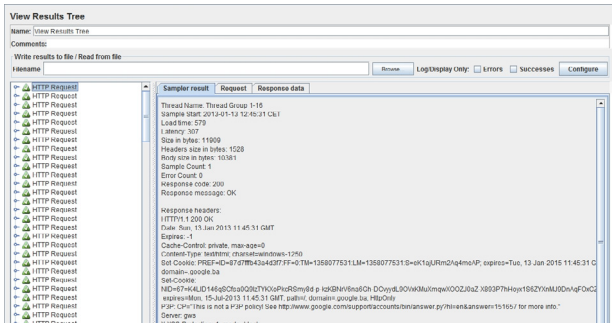
- 服务器名 (Server Name) 或 IP
- 路径 (Path)

定义完这个后，测试就准备好了，但我们通常需要一些测试报告。在 JMeter 中，我们称这种组件为监听器。因此，在这个测试计划中，加上一个监听器：



你的全部的请求响应结果都将会显示在这里。按 :Ctrl + R 开始运行这个测试。如果打开结果视图窗口 (View Results Tree)，你可以看到实时的运行状态。

运行完后，你可以再按：Ctrl + E 来清除旧的结果，并重新按 Ctrl + R 来重新启动一次新的测试。最后，你可以看到类似以下的结果：



正如我在一开始说的那样，这是一个最基本的 HTTP 负载测试计划。但 JMeter 提供的功能却远非如此简单，它能够通过建立大量的、你需要学习的各种选项来完成各种测试案例。我强烈推荐大家使用这个工具来进行各类测试。当然，这个工具用自定义插件来扩展也很简单。

AtlantBH 开发了一套 JMeter 的扩展插件，其中包括：

- OAuth Sampler
- REST Sampler
- JMS Tools (Java Request Sampler)
- JSON 到 XML 转换器
- XML Format Post-processor
- HDFS Operations Sampler
- HBase Scan Sampler
- HBase RowKey Sampler
- Hadoop Job Tracker Sampler
- HBase CRUD Sampler
- JSON utils (JSON Path Assertion, JSON Path Extractor, JSON Formatter) ■

如果说最近的 Java 暴露出来的安全缺陷能给我带来什么警示，那就是现在是到了 Oracle 公司重写这种语言的时候了。

专题链接：

http://developer.51cto.com/art/201301/378937.htm

渐进式吸引：让潜在用户在实践中了解产品、完成转化

编者按

所谓“渐进式吸引”，也不是什么前无古人后无来者的新鲜东西，作者将这个概念从一些实践案例中提取出来，归纳为一种产品设计和推广思路，蛮值得参考借鉴的。

所谓“渐进式吸引”，也不是什么前无古人后无来者的新鲜东西，作者将这个概念从一些实践案例中提取出来，归纳为一种产品设计和推广思路，蛮值得参考借鉴的。走起叭。

用户访问你的产品推广页面时可以看到些什么内容？通常也就是功能特色啊截屏啊口碑评价啊这些，外加一个或多个行为召唤 (call to action) 操作。这类传统的产品介绍页面一般都会面临两个挑战：首先，它必须让潜在用户对你的产品产生足够的兴趣，并进行注册或下载；第二，它需要在一定程度上让这些潜在用户对产品的运作方式及使用方法有一个大致的了解，使他们在正式使用产品的时候更容易上手。

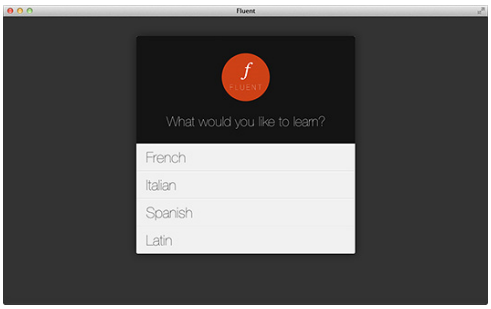
不妨试着将这两方面结合起来，引导潜在用户直接在产品介绍页面中通过某种方式试用产品。

让他们逐渐对产品的功能特色及使用方法产生认知，从而激发他们自主的执行下一步动作，例如注册或下载——我（英文原文作者）将这个过程称为“渐进式吸引”。

产品实例

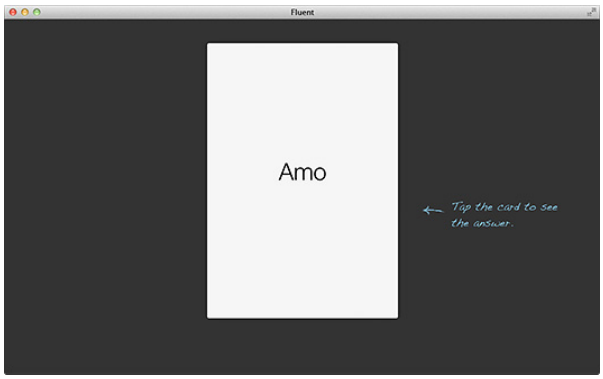
我们来看一个实际案例叭：我设计了一款 iPhone 上的抽认卡应用，名字叫作 Fluent，它的特点就是可以根据用户的实际水平，帮助他们重点记忆那些相对生疏的词汇。用户不需要做复杂的学习目标设置，打开应用就可以开始学习单词。其中的具体原理不在这里多做解释了，这本是一个非常简单易用的产品概念，但它在投放市场后并没有很好的引起用户的关注。（插一句，我们曾经发过这位作者的另外一篇文章，“将产品在移动应用市场中推向成功的十点建议”，他在其中也谈到了推广这款 Fluent 时遇到的一些问题；非常不错的一篇文章，再次推荐。另外，这种结合自己的产品案例撰写经验心得文章的做法，也不失为一种推广技巧　译者 C7210）。

为此，我专门制作了一个小站点，用来对 Fluent 进行介绍和推广。大家可以在下图中看到，这个页面当中没有任何功能介绍或是推广方面的内容，它只是询问用户希望学习哪种语言，包括法语、意大利语、西班牙语和拉丁语。

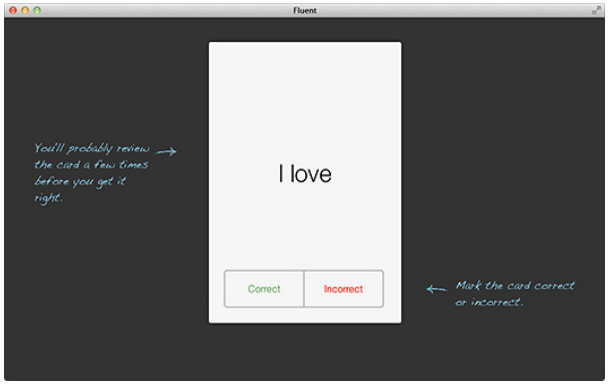


当用户选择了某种语言，譬如拉丁语，界面会通过动画效果滑入一张卡片，上面写有一个拉丁语的单词，同时在卡片旁边输出很简短的介绍文案，引导用户点击卡片，查看这个单词的解释。

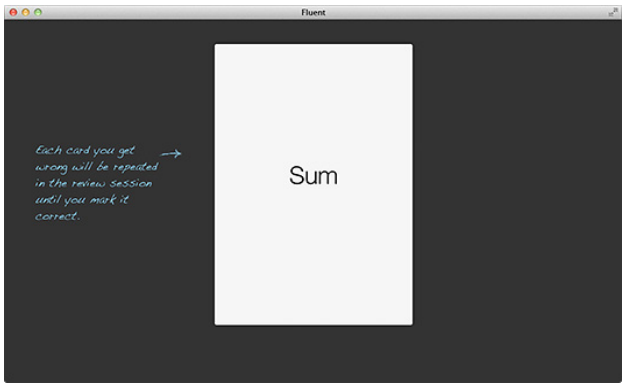
渐进式吸引 :让潜在用户在实践中了解产品、完成转化 II



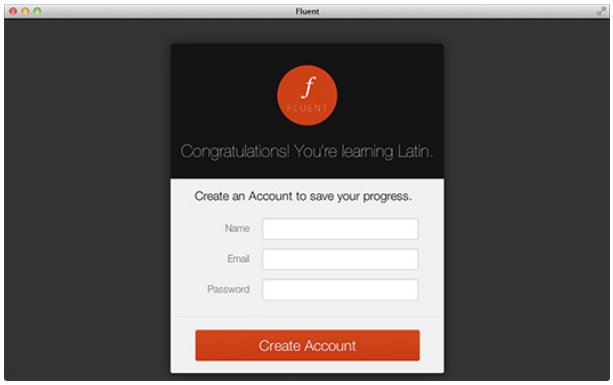
卡片被点击之后会翻转过来,这样用户就可以看到背面的单词解释了。同样,这里也会有提示文案,告诉用户在他们真正记住了这个词之前,系统还会再输出几次,以增强记忆。



用户需要对这个答案是否符合自己的记忆做出确认,即点击“正确”或“错误”;然后一张新的记忆卡会出现,同时,提示文案会告诉用户,如果他对之前那张卡片上的单词记忆有误,那么这个单词会在下一轮复习当中再次出现。



就这样循环下去…用户成功的记住了三个单词之后,就可以看到一个新的界面,标题是“恭喜!你已经走上拉丁语的学习之路了”,下面则是注册表单。



这里有个细节需要注意。你可以看到,我并没有在这个界面里显性的怂恿用户立刻去下载应用,而是提示他们通过创建账户的方式保存之前的学习进度。这样做可以让这些潜在用户了解到之前在试用和学习的过程里花费的时间与精力不是没有意义的——让用户感觉到他们的付出是能产生回馈与回报的,这很重要;这种“替用户考虑”的方式可以更加有效的提升转化率。

这就是我对“渐进式吸引”这个概念的实践方式——创建一个与自己产品相关的简易 Web 应用,在首页通过简单直白而符合目标用户核心需求的文案引导他们直接“试用”,在这个过程中“随需随到”的对产品功能特色及使用方式进行简要介绍;在这些潜在用户通过亲身实践对产品产生了渐进式的认知之后,再引导他们进行转化。■

原文未完,请参考

http://developer.51cto.com/art/201301/377781.htm

如何提高工作成效？克服对负面情绪的恐惧

你也许不知道，由于像费力克斯（当然还有你和我）这样拥有这种问题的人越来越多，在帮助人们如何变得更加有效率方面已经形成了一个完整的产业链。

1 月 28 日消息，《Fast Company》杂志网络版刊登了作家霍华德·雅各布森 (Howard Jacobson) 的一篇文章，文章分析了为何很多聪明、有创造性又上进的人却总是浪费大量时间在低价值、琐碎费时的事情，无法专注于高价值、决策性的“大图景”类工作的真正原因，那就是对于负面情绪的恐惧。

全文如下：

我的一名客户费力克斯（化名）抱怨每周的大量时间都被“浪费”掉了，作为一名精力充沛、目标明确的企业家，他却总是发现自己在做一些技术性的、行政类的工作，虽然他内心很明白，自己应该做的是那些真正能帮助公司发展前进的“高价值类”工作。

更让人困惑的是，其实费力克斯拥有可以做这些行政和技术类工作的员工，并且为部分工作雇佣了可靠的外包公司——他到底在想些什么？为什么他似乎“控制不住”地抛开那些他热爱去做、并且只有他能做的高价值工作而去做那些普通员工就可以做也应该由他们来做的事情？

他要怎么样才能将这种形式逆转过来，去实现自己的真正价值？

“如何高效工作”产业链

你也许不知道，由于像费力克斯（当然还有你和我）这样拥有这种问题的人越来越多，在帮助人们如何变得更加有效率方面已经形成了一个完整的产业链。在这方面，各种各样的工具和项目如软件、提建议的博客、培训、硬件、应用、书籍等等，应有尽有。

这些工具和项目无外乎围绕以下方面做文章，相信你也曾都看到过：

列出问题清单

列出需要优先解决的问题

优先解决最重要的问题

然而，对于客户费力克斯的例子来说，这些似乎都不是问题，他拥有很强的加强工作成效的意愿，懂得这些怎样“如何高效工作”的原则和工具，但为什么这一切似乎都不起作用？

本质上的自我破坏：提高成效就像节食减肥

当我们把“提高工作成效”这个目标变成“减肥”时，一切问题似乎都变得更加清楚。

如何提高工作成效？克服对负面情绪的恐惧 II

我们知道“减肥”界永不停歇的一个辩论是“何为最佳减肥餐饮？”，尽管这个话题一直有争议，但相信没有任何一个节食顾问或者健康专家会推荐苏打汽水、糖果、饼干、甜甜圈、油腻的汉堡和白面包。然而，大多数节食减肥者，虽然对于这些原则都了然于心，依然反反复复地偷吃这些会毁掉自己大目标的食物。我的客户费力克斯，正如这些节食减肥者，他们每时每刻的小决定破坏了他们最终的大目标。

为什么？

什么在真正激励行动？

在《习惯的力量》（The Power of Habit）一书里，查尔斯·迪黑格（Charles Duhigg）描述了一种将人从日常习惯中拉出来的力量，这种力量对人来说必须是一种“奖励”，只有我们认为去做这件事情的话我们可以得到奖励——得到一阵能量、一种与他人相联系的正面情感、一些开心或愉悦的情绪，或者是可以避免负面的情感和情绪——我们的内心就会驱使自己去做这件事情，哪怕这件事情不是我们通常喜欢做的或者习惯做的，毕竟，“趋利避害”是人和动物都有的天性。

在没有这种“奖励”的情况下，我们一般只会遵循自己的习惯，而且，每次对习惯的延循，都是在无形中对这种习惯的一次加深，久而久之，有些行为甚至会进化成为一种近乎“本能”的反应。

我们当然可以尝试去跳出这种“习惯圈”（habit loop），所以我们会列出“新年要做的事”“近期目标”等等，但是，研究证明，意志力其实是一种有限的资源，是一块很容易就松懈的肌肉，而我们想要逃脱的习惯却无比顽固。换句话说，在你的“长期目标的意志力”与“短期习惯下的松懈”这场战斗中，后者总是轻松获胜。

时间操纵是无效的

在“搞定你的工作”这个行业里几乎所有的策略都包含了一些“时间操纵”的成分，也就是用一些小伎俩进行自我欺骗，以达到驱使我们去做在特定时刻我们本不想去做的事情的目的。但这些不过是骗人的把戏，短期内或许可以成功一次两次，到头来当我们觉察到了这些诡计的本质，我们依然会重回旧路。

这就如同在一个腐朽破旧的引擎外边套上一个崭新的汽车车身一样，想要在一个旧的坏的习惯的基础上建立一个新的健康的习惯，就必须解决最根本的问题，否则只是换汤不换药。

所以，按照常规的建议，我原本可以向客户费力克斯建议：每天早上抽出两个小时做最重要的事情。设立一个电子日程提醒，甚至可以在自己的桌子上摆一个提醒闹钟，告诫自己在这两个小时内不要去检查邮件、不要写网页代码、不要支付账单、不要查看 Facebook。

这个建议在短时间内可能有效。

但最终，有件事情注定会发生，那就是费力克斯有天会发现自己在那两个小时内无事可做。

他把所有有价值的工作都做完了，他坐在那里，对自己很满意，这种状态持续了四秒钟。

然后他就又开始有了那些感觉。

具体什么感觉我也不清楚，人在空虚无聊的时候会想些什么呢？ ■

原文未完，请参考

http://developer.51cto.com/art/201301/378973.htm

Windows 8开发之动态磁贴辅助磁贴

作者 /Simple Simle

磁贴在系统的开始菜单中代表着应用，动态磁贴可以向用户显示新的、重大的、定制的内容。Tile 通知是一种固定格式的 XML，其中包含文字和图片内容。

在应用程序清单中，必须包含一张正方形的的 logo，如果 应用程序也想使用宽模版 logo，也需要在清单中注明。如果你的应用中同样支持宽 tile，强烈建议你预加载 方形和宽形 在预加 载的 xml 中，从而无论开始菜单中的 tile 是方形或者宽形的 都可以接收通知。

以下是微软提供的磁贴模版：

磁贴和磁贴通知：http://msdn.microsoft.com/zh-cn/library/windows/apps/hh779724.aspx

磁贴模板目录： http://msdn.microsoft.com/zh-cn/library/windows/apps/hh761491.aspx

1. 动态磁贴

使用 Windows.UI.Notifications 命名空间下的 TileUpdateManager 类，创建用于更改和更新启动菜单图块的 TileUpdater 对象。此类提供对系统提供的平铺模板 XML 内容的访问，以便您可以自定义用于更新您平铺的内容。 具体模版的 XML 内容，可以根据微软的模版，进行修改。

另外我们可以下载 NotificationsExtensions 文件，将该文件引用到我们的项目中，使用该文件中提供的类和方法来创建我们的动态磁贴，这个和前面不同的地方是，我们可以不使用 XML 模版进行设置磁贴的模版，而是通过具体的模版类，给模版类的属性赋值，实现动态磁贴。下面的代码实现了，最近五个动态磁贴之间的切换，该模版显示的样式为：

宽磁贴：左侧是一个较小的图像，右侧上面是第一行上较大文本的标题字符串，下面是四行四个常规文本的字符串。文本不自动换行。

窄磁贴：上面是一个较大文本的标题字符串，下面是一个最多可包含三行自动换行常规文本的字符串。注意：动态磁贴必须要在实体机器上，在模拟器中是无法显示动态磁贴的效果。

2. 辅助磁贴（二级磁贴）

辅助磁贴使用户能够将 Windows 应用商店应用的特定内容和深层链接—对固定应用内部一个特定位置的引用—发送到“开始”屏幕上。辅助磁贴使用户能够使用好友、新闻源、股票报价和其他对其很重要的内容来个性化“开始”屏幕体验。创建辅助磁贴的选项最常在 UI 中显示为“附到开始菜单”选项。固定内容也就是为它创建辅助磁贴。此选项常常显示为应用栏上的一个标志符号。通过触摸或单击来选择辅助磁贴，会启动到父应用，以突出一种以固定内容或联系人为中心的体验。只有用户才可以固定辅助磁贴；应用不能在未获得用户许可的情况下以编程方式固定辅助磁贴。用户还可以通过“开始”屏幕或通过父应用，对辅助磁贴进行显式删除控制。

1). 在固定辅助磁贴前，用户必须确认，要求对此进行确认的弹出窗口应当显示在调用固定请求的按钮旁边。■ 本文源代码请访问原文：

http://developer.51cto.com/art/201302/380283.htm

数据结构中二叉树的基本操作

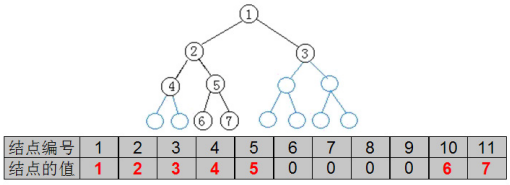
二叉树的基本操作，可能包括：

创建，遍历，转化，复制，删除等。

遍历：前中后三种顺序的遍历，已经是各数据结构与算法教程的最基础内容，在此不重复。

创建：大多数据结构教程当中的二叉树创建程序，都是采用的递归方式，递归方式创建的二叉树与遍历的过程相似，所创建的二叉树，也是采用左右子节点方式，后续进行遍历操作十分方便。

转化：直觉上，最简单的二叉树存储方式其实是如下图的数组：



结点编号	1	2	3	4	5	6	7	8	9	10	11
结点的值	1	2	3	4	5	0	0	0	0	6	7

* 此图出自某高校数据结构 ppt，但实在难以查证是哪个学校，无法直接感谢，请谅解。

首先，提供个满二叉树大小的数组，然后其中数值按完全二叉树存储。

显然，此种顺序存储方法：第 i 号（这里编号指对应的完全二叉树的位序）结点的左右孩子一定保存在第 2i 及 2i+1 号单元中。

故此，为兼顾存储的直观与遍历等操作的方便，从顺序数组向左右子节点存储方式的转化也就十分重要。

1- 转化方法

分为几个步骤：

（1）准备原始数组

（2）分析数组中的有效值，对应二叉树节点非空；

（3）创建二叉树节点；

（4）计算除最后一层子节点外，构造节点间父子关系时的循环次数；

（5）构造二叉树节点间的父子关系；

（6）确实二叉树根节点；

主要代码：

（1）准备原始数组

```
// 原始数组
    int intBiTreeInit[ARR_COUNT];
    // 初始化原始数组至无效值
    for(int i=0;i<=ARR_COUNT-1;i++)
      intBiTreeInit[i]=NVALUE;
     // 本 if 条件确保 ARR_COUNT 是否是的乘方 -1
    if(0==(ARR_COUNT & (ARR_COUNT+1)))
    {
      for(int i=0;i<=ARR_COUNT-1;i++)
        intBiTreeInit[i]=2*(i+1);
    }
    else
      return RET_ERR;
    // 使最后两数为无效值
    intBiTreeInit[ARR_COUNT-1]=NVALUE;
    intBiTreeInit[ARR_COUNT-2]=NVALUE;
```

（2）分析数组中的有效值

开始获得数组中有效值位置

数据结构中二叉树的基本操作 II

```
int intRel=0;
int intArr=0;
for(intArr=0;intArr<=intCount-1;intArr++)
{
    if(elemArr[intArr]!=elemNValue)
    {
        intRel++;
        vecIntEffPos.push_back(intArr);
    }
    }
```

（3）创建二叉树节点

```
// 数组中有效值对应创建节点
// 同时初始化父子节点为 NULL
for(intArr=0;intArr<=intRel-1;intArr++)
{
        pBiTreeTemp=(PBiTreeNode)
malloc(sizeof(BiTreeNode));;
        if(NULL==pBiTreeTemp)
// 判断是否有足够的内存空间
        {
            cout<<"Memory alloc failure"<<endl;
            return RET_ERR;
        }
        // 将有效值赋予节点
        pBiTreeTemp->BiTreeData=elemArr[vec
IntEffPos[intArr]];
        // 初始化左右子节点为 null, 便于后续的
遍历
        pBiTreeTemp->leftChild=NULL;
        pBiTreeTemp->rightChild=NULL;
        // 先存节点值
        vecPBiTree.push_back(pBiTreeTemp);
```

（4）计算除最后一层子节点外，构造节点间父子关系时的循环次数

```
// 生成父子关系时最后一层不必遍历 , 故
理论循环上限可优化
int intSubLast=0;
intSubLast=intCount-(intCount+1)/2;
```

（5）构造二叉树节点间的父子关系

```
for(intArr=0;intArr<=intSubLast-1;intArr++)
{
    // 左右节点若存储有效值则同时创建父子
关系
    if(elemArr[intArr*2+1]!=elemNValue)
        vecPBiTree[intArr]->leftChild=vecPBiT
ree[intArr*2+1];
    if(elemArr[intArr*2+2]!=elemNValue)
        vecPBiTree[intArr]->rightChild=vecPBi
Tree[intArr*2+2];
```

（6）确实二叉树根节点

```
pBiTree=vecPBiTree[0];
```

转化为左右子节点方式存储后，则各种遍历操作按大多数教程的常规方式处理即可，如前序遍历函数：

完整程序，请见附件文件。

http://files.cnblogs.com/vbspine/cnsDSExec.rar

* 上述程序在 Windows7x64，VS2008 环境编译运行通过

■本文未完，更多代码请参考原文：

http://developer.51cto.com/art/201301/379141.htm